AMENDMENTS TO THE SPECIFICATION

On page 4, lines 6-11, please replace the paragraph as follows:

--One of the difficulties in providing isolation between virtual private servers within a single host computer involves resource ownership. In UNIX® and related operating systems, certain system resources, such as processes and files, are owned by users or ~~group~~ groups of users. Each user is assigned a user identifier (UID) by which the user is identified in the operating system. In some cases, a group of users may be assigned a group identifier (GID).--

On page 6, lines 9-16, please replace the paragraph as follows:

--However, if a user of each virtual process ~~was~~ were given full super-user privileges, a super-user of one virtual process could access the files of a user of another virtual process. Similarly, a super-user of one virtual process could terminate the processes associated with a user of another virtual process. Indeed, a super-user of one virtual process could obtain exclusive access to all system resources, effectively disabling the other virtual processes. Clearly, allowing a user of each virtual process full super-user privileges would seriously compromise system security, entirely removing the illusion that the virtual processes are running on dedicated host computers.--

On page 12, lines 12-17, please replace the paragraph as follows:

--Accordingly, one aspect of the present invention relates to a system and method for associating identifiers with virtual processes, as described immediately below. Thereafter, a system and method are described for virtualizing resource ownership in ~~an~~ a computer operating system including multiple virtual processes. Finally, there is provided a detailed description of a

system and method for virtualizing super-user privileges in a computer operating system including multiple virtual processes.--

On page 15, lines 8-14, please replace the paragraph as follows:

--In one embodiment, rather than starting a separate system initialization process 107 for each virtual process 101, a custom initialization process is started. In this embodiment, all processes that are [[a]] members of a specific virtual process 101 are descended from the associated custom initialization process, and are associated with the virtual process 101 with which the custom initialization process is associated. The exact functionality included in the custom initialization process is a design choice that can be made by, for example, a system administrator.--

On page 20, lines 12-18, please replace the paragraph as follows:

--As illustrated in FIG. 3, one of the difficulties in providing isolation between virtual processes 101 (e.g., virtual private servers) within a single host system 300 involves resource ownership. In UNIX® and related operating systems 117, certain system resources, such as processes 301 and files 303, are owned by users or ~~group~~ groups of users. Each user is assigned a user identifier (UID) 305 by which the user is identified in the operating system 117. In some cases, a group of users may be assigned a group identifier (GID) 307. The UID 305 and GID 307 are sometimes referred to herein as "owner identifiers."--

On page 25, lines 7-11, please replace the paragraph as follows:

--After the virtual process 101 is determined, the system call wrapper 111 encodes 611 an indication of the virtual process 101 (e.g., the VPID 203) within the UID 305. The wrapper 111 then associates 613 the ~~calling process 301~~ resource with the modified UID 305. In one implementation, this is accomplished by executing the system call 115 within the wrapper 111, specifying the modified UID 305.--

On page 28, lines 16-21, through page 29, lines 1-8, please replace the paragraphs as follows:

--After the translation data structure 907 is updated, the wrapper 111 associates the ~~calling process 301~~ resource with the alternative UID 901. This is accomplished, in one embodiment, by executing the system call 115, specifying the alternative UID 901.

FIG. 9 provides an example of the above-described technique. Suppose that a process 301 having a PID 201 of ~~3942~~ 1847 attempts to execute the UNIX® setuid() system call 115 with a specified UID 305 of 1. As illustrated, the system call wrapper 111 intercepts the call 115 and uses the virtual process table 127 to determine the virtual process 101 (e.g., VPID 203) associated with the calling process 301.

The system call wrapper 111 then selects an alternative UID 901 (e.g., 1003) from a set 903 of available UIDs 305. Thereafter, the wrapper 111 creates an association 905 in the translation data structure 907 between the UID 305 specified in the call 115 (e.g., 1), the alternative UID 901 (e.g., 1003), and the VPID 203 (e.g., ~~1~~ 2). Once the translation data structure 907 is updated, the wrapper 111 associates the calling process 301 with the alternative UID 901 by executing, for example, the system call 115.--

On page 30, lines 7-9, please replace the paragraph as follows:

--The wrapper 111 then accesses the translation data structure 907, looking up ~~a~~ an alternative UID 901 of 1003 and a VPID 203 of 2. As illustrated, an association 905 exists, revealing a UID 305 of 1, which is subsequently returned to the calling process 301.--

On page 31, lines 3-21, please replace the paragraphs as follows:

--However, if a user of each virtual process 101 ~~was~~ were given full super-user privileges, a super-user of one virtual process 101 could access the files 303 of a user of another virtual process 101. Similarly, a super-user of one virtual process 101 could terminate the processes 301 associated with a user of another virtual process~~.~~ 101. Indeed, a super-user of one virtual process 101 could obtain exclusive access to all system resources, effectively disabling the other virtual processes 101. Clearly, granting a user of each virtual process 101 full super-user privileges would seriously compromise system security, entirely removing the illusion that each virtual process 101 is running on a dedicated host computer.

As illustrated in FIG. 11, the present invention solves the foregoing problems, in ~~on~~ one embodiment, by designating a plurality of virtual super-users 1101, typically one per virtual process 101. A virtual super-user 1101 has many of the privileges of an actual super-user with respect to his or her own virtual process 101. For example, a virtual super-user 1101 can add and delete users of the virtual process 101, access files 303 of any user of the virtual process 101, terminate processes 301 associated with the virtual process 101, and the like. However, a virtual super-user 1101 cannot, for instance, add or delete users of other virtual processes 101, access the files 303 of users of other virtual processes 101, or terminate the processes 301 associated with other virtual processes 101.--

On page 32, lines 5-8, please replace the paragraph as follows:

--~~As previously noted, a~~ A UID 305 of zero is interpreted by UNIX® and related operating systems as the super-user UID 305. However, assigning a UID 305 of zero to each virtual super-user 1101 would result in the problems discussed above, since an actual super-user has unfettered access to all system resources.--

On page 32, lines 15-20, please replace the paragraph as follows:

--For instance, as shown in FIG. 11, a VPID 203 of 1 is encoded within the upper 16 bits of the VSUID 1103, resulting in a VSUID 1103 of 0x00010000. Likewise, a VPID 203 of 2 results in a VSUID 1103 of 0x00020000. Finally, a VPID 203 of 3 results in a VSUID 1103 of ~~0x0003000~~ 0x00030000. Of course, those skilled in the art will recognize that the VSUID 1103 may be encoded in various ways without departing from the spirit and scope of the invention.--

On page 33, lines 6-21, through page 34, lines 1-13, please replace the paragraphs as follows:

--Consequently, as shown in FIG. 12, selected system calls 115 are intercepted for performing operations requiring actual super-user privileges, which are nevertheless desirable for a virtual super-user 1101 to perform in the context of his or her own virtual process 101. For example, system calls 115 are intercepted that operate on files 303, e.g., open(), creat(), link(), unlink(), chdir(), fchdir(), symlink(), readlink(), readdir(), access(), rename(), mkdir(), rmdir(), truncate(), and ~~ftrunate()~~ ftruncate(). Of course, those skilled in the art will recognize that the invention is not limited to any particular operating system 117 or terminology.

As noted above, a normal user is typically restricted from opening, deleting, renaming, etc., a file ~~301~~ 303 owned by another user. However, a virtual super-user 1101 should appear, in most respects, to be an actual super-user for operations pertaining to his or her own virtual process 101.

Thus, in one embodiment, if a system call 115 is "made" by a virtual super-user 1101 (i.e., by a process 301 owned by a virtual ~~super-user~~ super-user 1101) and pertains to the virtual process 101 of the virtual super-user 1101, then actual super-user privileges are temporarily granted to the virtual super-user 1101 for purposes of the system call 115. This may be accomplished, in one embodiment, by executing an appropriate function, e.g., setuid(), to assign a UID 305 of zero or other designation of super-user privileges to the calling process 301. After the system call 115 is executed, the super-user privileges may be withdrawn by executing the same function to restore the VSUID 1103.

Whether the system call 115 was made by a virtual super-user 1101 may be determined by checking whether the owner of the calling process 301 has a VSUID 1103. Of course, if the system call 115 was not made by a virtual super-user 1101, the wrapper 111 preferably disallows execution of the system call ~~111~~ 115. For instance, the wrapper 111 may generate an error message, indicating a privilege violation. Alternatively, the wrapper 111 may simply allow the system call 115 to proceed without granting actual super-user privileges, resulting in the operating system 117 disallowing execution of the system call 115, since the VSUID 1103 does not convey actual super-user privileges.--

On page 35, lines 3-7, please replace the paragraph as follows:

--As shown in FIG. 12, suppose that a process 301 owned by a virtual super-user 1101 attempts to execute the open() system call 115 in order to open another user's file 303, which is nevertheless associated with the virtual process 101 of the virtual super-user 1101. The virtual process 101 (e.g., VPID 203) may be determined, in one embodiment, by referencing the virtual process table 127 using the PID 201 of ~~"3542."~~ "3942."--

On page 36, lines 3-10, please replace the paragraph as follows:

--Other system calls 115 that may be intercepted include system calls 115 for terminating a process 301. In UNIX®, the kill() system call 115 allows a user to terminate one or more processes 301. For example, executing the kill() system call 115 with a specified process 301 (e.g., PID 201) terminates that process 301. Executing the kill() system call 115 with an argument of -1 results in the termination of all of the user's processes 301. An argument of less than -1 results in the termination of all of the processes 301 associated with a group (e.g., GID 307, ~~taken from~~ where the GID value is equal to the absolute value of the argument).--

On page 36, lines 20-21, through page 37, lines 1-9, please replace the paragraphs as follows:

--Where a virtual super-user 1101 attempts to terminate a specific process 301 associated with his or her virtual process 101, the wrapper 111 proceeds as discussed above with reference to FIG. 12. In other words, the wrapper 111 grants temporary actual super-user privileges ~~101~~ to the calling process 301 and allows execution of the system call 115.

However, as shown in FIG. 13, where the system call 115 specifies a negative parameter, the wrapper 111 proceeds differently. Since the powers of virtual super-user 1101 should be

limited to his or her virtual process 101, a kill() system call 115 with an argument of -1 results only in the termination of processes ~~101~~ 301 associated with the virtual process 101. Thus, in one embodiment, a kill(-1) system call 115 "pertains" to the virtual process 101 by definition.--

On page 37, lines 15-21, through page 38, lines 1-3, please replace the paragraphs as follows:

--Likewise, in the case of an argument of less than -1, denoting a GID 307, the wrapper 111 cycles through all of the processes 301 associated with the virtual process 101 of the virtual super-user 1101 and determines whether each such process 301 corresponds to the specified group (e.g., GID 307). If so, those processes 301 are terminated in the manner discussed above.

As an example, as shown in FIG. 13, suppose that a process 301 is associated with a virtual process 1 (e.g., having a VPID 203 of 1). The process 301 is owned by a virtual super-user 1101 by virtue of the VSUID 1103 (e.g., 0x00010000), and pertains to the virtual process 101 by definition. Accordingly, the wrapper 111 grants temporary actual super-user privileges to the calling process 301 by executing the system call 1201.--

On page 38, lines 19-21, through page 39, lines 1-2, please replace the paragraph as follows:

--In one embodiment, a virtual super-user 1101 is prevented from executing a system ~~calls~~ call 115 for which actual super-user privileges are required by simply not intercepting the call 115. Since the VSUID 1103 is not a super-user UID 305, the operating system ~~115~~ 117 will automatically reject an attempt by a virtual super-user 1101 to execute, for example, the ~~insertmod()~~ insmod() call 115.--

On page 40, lines 17-21, through page 41, lines 1-3, please replace the paragraph as follows:

--~~If~~ In view of the foregoing, the present invention offers numerous advantages not available in conventional approaches. For example, super-user privileges are virtualized in an operating system 117 including multiple virtual processes 101, such that a virtual super-user has the power to perform traditional system administrator functions with respect to his or her own virtual process 101, but is unable to interfere with other virtual processes 101 or the underlying operating system 117. Thus, each virtual process 101 can have a virtual super-user 1101, while preserving the illusion that the virtual processes 101 are running on dedicated host machines.--